Copyright 2001

by

Diep Thi Ngoc Nguyen
ACKNOWLEDGMENTS

This thesis is the result of my strong commitment to study a new field that brought me so many challenges and valuable experiences. It was Dr. Chow’s two operating system classes that inspired me and led me to start this thesis. I would like to express my sincere gratitude and appreciation to my advisor Professor Randy Chow for all his valuable guidance and encouragements. I greatly appreciate my committee members Dr. Jonathan Liu, Dr Richard Newman and Dr. Douglas Dankel for their advice and assistance. Thanks also go to all members of the network security seminar for sharing with me valuable comments and knowledge. I would like to thank the faculty and staff of the Computer and Information Science and Engineering department at the university of Florida for all their assistance during my stay at the department.

This work could not be done without the unconditional love and support from my family, to whom my thesis is dedicated. My love for my family provides endless motivation and inspiration to help me overcome obstacles and fulfill my goals.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vii</td>
</tr>
<tr>
<td>CHAPTERS</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivations in Developing Role-Based Access Control</td>
<td>2</td>
</tr>
<tr>
<td>1.2 What Is the Best Environment for Using RBAC</td>
<td>7</td>
</tr>
<tr>
<td>1.3 RBAC Terms and Concepts</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Roles and Related Concepts</td>
<td>10</td>
</tr>
<tr>
<td>2 KNOWN APPROACHES TO DESIGN AND IMPLEMENTATION OF RBAC</td>
<td>11</td>
</tr>
<tr>
<td>2.1 A Language for RBAC Systems</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Generalized RBAC Access Control (GRBAC)</td>
<td>16</td>
</tr>
<tr>
<td>2.3 Administrative Models for RBAC</td>
<td>18</td>
</tr>
<tr>
<td>2.4 An Object-Based Implementation of Role Based Access Control</td>
<td>19</td>
</tr>
<tr>
<td>2.5 Conclusion</td>
<td>21</td>
</tr>
<tr>
<td>3 EXISTING RBAC SYSTEMS</td>
<td>23</td>
</tr>
<tr>
<td>3.1 RBAC in Business and Commercial Environment</td>
<td>24</td>
</tr>
<tr>
<td>3.2 RBAC for the Network Environment</td>
<td>27</td>
</tr>
<tr>
<td>3.2.1 RBAC for Intranet Security</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2 RBAC for Network Enterprises</td>
<td>29</td>
</tr>
<tr>
<td>3.2.3 RBAC for Different Types of Internet Documents</td>
<td>30</td>
</tr>
<tr>
<td>3.2.4 RBAC for WWW Servers</td>
<td>32</td>
</tr>
<tr>
<td>3.3 RBAC in Heath Care Environment</td>
<td>34</td>
</tr>
<tr>
<td>3.4 RBAC in Multi-Level Secure Systems</td>
<td>35</td>
</tr>
<tr>
<td>3.5 Other Role-Based Access Control Models</td>
<td>36</td>
</tr>
<tr>
<td>3.6 Conclusion</td>
<td>37</td>
</tr>
</tbody>
</table>
4 EXTENDING RBAC MODELS WITH NEW CONCEPTS

4.1 Control Information Flow in Role-Based Access Control
4.2 RBAC for Workflow Management
4.3 Extending RBAC Model with States
4.4 RBAC and Session Management
4.5 RBAC and Type-Enforced Systems
4.6 Conclusion

5 AN EXAMPLE OF APPLYING RBAC CONCEPTS IN THE DESIGN AND
SPECIFICATION OF AN ACCESS CONTROL SYSTEM

5.1 Role Creation
5.2 System Component Definitions
5.2.1 Data Elements
5.2.2 System Requirement Specifications
5.3 Suggested System Design
5.3.1. System Management
5.3.2. Role Server
5.4 Conclusion

6 PROPOSED EXTENSIONS TO THE CURRENT NIST RBAC/WEB SYSTEM

6.1 Requirement Specifications for Core RBAC
6.1.1 Core RBAC System Administrative Functions
6.1.2 Supporting System Functions
6.1.3 Review Functions
6.2 Design Specification
6.3 System Modules
6.3.1 RBAC Database
6.3.2 Administration Module
6.3.3 System Administration API Module
6.4 Advanced Review for RBAC/Web System
6.5 Assumptions
6.6 Rationale

7 CONCLUSION

REFERENCES

BIOGRAPHICAL SKETCH


LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 An example of RBAC hierarchy</td>
<td>5</td>
</tr>
<tr>
<td>2 Grammar for RDL</td>
<td>13</td>
</tr>
<tr>
<td>3 Implementing RBAC with layered objects</td>
<td>21</td>
</tr>
<tr>
<td>4 A layered RBAC structure</td>
<td>30</td>
</tr>
<tr>
<td>5 Illegal information flow</td>
<td>39</td>
</tr>
<tr>
<td>6 Legal information flow</td>
<td>41</td>
</tr>
<tr>
<td>7 A role-based perspective of workflow</td>
<td>42</td>
</tr>
<tr>
<td>8 An example of state transitions graph</td>
<td>46</td>
</tr>
<tr>
<td>9 Access control process in a type enforcement system enforcing RBAC</td>
<td>50</td>
</tr>
<tr>
<td>10 The organizational structure of the CISE department</td>
<td>52</td>
</tr>
<tr>
<td>11 Suggested system architecture</td>
<td>57</td>
</tr>
<tr>
<td>12 RBAC/Web use</td>
<td>63</td>
</tr>
<tr>
<td>13 Table1 The main components of RBAC/Web</td>
<td>33</td>
</tr>
</tbody>
</table>
Abstract of Thesis Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Master of Science

A STUDY ON ROLE-BASED ACCESS CONTROL

By

Diep Thi Ngoc Nguyen

December 2001

Chairman: Dr. Yuan-Chieh Chow
Major Department: Computer and Information Science and Engineering

This thesis represents a broad investigation on one of the most flexible, expressive yet simple and easy to manage access control mechanisms, namely role-based access control (RBAC). Even though RBAC has been extensively researched and developed only during the last decade, it has been proved to be among the most promising access control mechanisms. Our study shows that RBAC has been implemented in one way or another in many fields of information science and technology, especially in distributed network environments. The attention that has been drawn to RBAC is justified by numerous advantages of RBAC over other existing access control models. The main purpose of this thesis is to draw a big picture representing the development process of RBAC over the last decade, analyze the existing RBAC models and the main issues of their design and implementation as well as propose possible extensions/improvements to RBAC models. We demonstrate the feasibility of RBAC concepts through an example based on a real access control policy. Since RBAC is a multi-dimensional concept
ranging from very simple at one end to quite sophisticated at the other, instead of a single model, a family of RBAC models is defined to accommodate all of these variations. The ultimate goal of much RBAC research is to develop complete, well-functioning RBAC systems which can deploy the main features of RBAC to make them easily adaptable to different access control requirements.
CHAPTER 1
INTRODUCTION

In today’s computing world, when people all over the world enjoy every aspect of information technology, information security has become a very urgent and complex problem. Unauthorized accesses to protected information can lead to unimaginable damages. Providing a secure, flexible, policy neutral and yet simple to administrate access control model is the ultimate aim of many security researchers and engineers. Different access control models, which offer certain advantages, have been proposed, but a complete, well-studied model is still to be built. One of the promising approaches to solving access control tasks is the role-based access control model.

During the last ten years, role based access control model has attracted a great deal of attention, mainly because of its advantages over other existing access control models. National Institute of Standards and Technology (NIST) [San00] has defined a role-based access control model, which serves as a standard for other role based access control models. Yet this is still a starting point, much interesting and challenging work remains to be done. The purpose of this thesis is to investigate the RBAC, analyze existing RBAC models and the features that they provide as well as their design and implementation issues (Chapter 2 and Chapter 3). Since we find that the following concepts: information flow, work flow, session and state, play a very important role in making access control decisions, we analyze the impact of these concepts in RBAC in Chapter 4. In Chapter 5 we demonstrate how to apply RBAC concepts in the design and
specification of a RBAC system using an access control policy of the CISE department at the University of Florida as an example. At the time this paper is being written one of the most advanced RBAC model, which implements most of the RBAC components, is the RBAC/Web system for Web servers. The package is developed by a RBAC team at NIST which implements many of the main concepts of RBAC such as role hierarchies, static and dynamic separation of duties and role inheritance. In this thesis we propose an extension to the existing RBAC/Web, namely using Java, JDBC, Sybase and SQL to implement a RBAC/Web system API, and add advanced administrative functions to the system that haven’t been implemented so far in available RBAC products (Chapter 6). Chapter 7 will conclude our work with the list of open problems in this area.

1.1 Motivations in Developing Role-Based Access Control Mechanism

The main entities in access authorization are active subjects, which access some protected passive objects. One of the main issues in access control, especially in a distributed environment, where users and resources are dispersed, is the large number of subjects and objects that need to be managed. To solve this problem the notions of “group,” “role,” “compartment,” “level,” etc. have been introduced. Subjects with the same access rights can be grouped together into a group. Similarly, objects can be classified into different levels for security purposes (such as top-secret, secret, confidential, etc.). Moreover, access permissions can be gathered into sets of privileges to simplify the access control management. Simplifying security policy administration and at the same time improving the expressive power of the access control models to provide flexible, customized policies was and still is the main purpose of an ideal access control model.
The effectiveness of traditional access control models (Mandatory Access Control and Discretionary Access Control) is greatly challenged by the emerging distributed systems, which have much greater numbers of users, objects, roles, and program components. New modeling concepts and techniques have been extensively studied and among them RBAC models have shown their superior power in terms of expressiveness and flexibility over other existing models. A large number of RBAC models have been introduced to different types of distributed applications (commercial object systems, banking systems, government, and health care sectors), which greatly simplify the tasks of managing a large number of users (i.e. “active subjects”) or user groups and large numbers of access control permissions/system privileges or system/data objects.

In general, the concept of roles is much wider than the subjects in access control models. Roles can be used to describe the logical structure of any organization. For example, a department is composed of a chairman, associate chairman, faculty (professors, instructors), staff (secretaries, clerks), and students (undergraduate, graduate, postbac). A study by NIST [Fer93] has discovered a very close connection between the access control decisions and the roles that individual users take on as part of the organization. People are assigned to roles to fulfill some job functions, as users (subjects) who need permissions to carry some operations on protected objects. The power of RBAC lies in its ability to express access control policy in terms of the way in which administrators view organizations, i.e. in terms of the roles that individuals play within the organization. Most organizations have their own access control policy that must be maintained by the system administrators. But study [Bark97a] found that while many existing products do not meet all the unique access control needs of studied
organizations, RBAC models meet many needs and requirements of the government and commercial sectors including privacy of personal information, prevention of unauthorized access and distribution of financial assets or confidential information.

Roles have also been widely used in database management systems, security administration, technology, and business areas. Roles are defined and implemented in Oracle 7 as a part of the SQL3 standard. A number of other products also support some form of RBAC. For example Microsoft Windows NT operating system has implemented four different user roles: administrator as the super user, operator who has subsets of administrator privileges appropriate for a certain job functions, user who has no need to perform system and/or network administration tasks, and guest with the least privilege.

One of the main advantages of RBAC is that RBAC changes very little over time since transactions are associated with roles, not users. Even though people often change their jobs, but the jobs do not change themselves. A faculty member’s duties include teaching, research, and departmental services are independent of the fact who the actual individual is. And many people can be in the same role and so have the same access privileges. RBAC policies are written for the job, not the person who performs it. So changes in personnel virtually have no effects on an RBAC policy. This characteristic greatly simplifies the security management, especially in large organizations.

Roles in RBAC can be hierarchical and thus support inheritance and delegation of access permissions. Access privileges can be passed down from a senior role (a role at a higher level in the role hierarchy) to a junior role (a child of the senior role in the role hierarchy). Access permissions can also be delegated from one role to another. A manager can let his employee act on behalf of him to perform some of his tasks. While
inheritance and permission delegation share some common semantic value, these are two different concepts. Inheritance is closely connected with the hierarchical structure of an organization and permission inheritance is allowed only within an organization. Permission delegation can happen in an interdomain environment.

RBAC reduces the number of subjects and associations to be managed, since many users may be assigned to similar roles. There are more than 40 000 students in the University of Florida but they all can be categorized as in Figure 1.

![Figure1: An example of RBAC hierarchy](image)

A role can enclose sets of permissions related to a job function or responsibilities within an organization which enables us to easily assign/collect/revoke the access permissions associated with a role. This is an important feature that is seldom provided by other object-based access control models.

RBAC allows both per-subject (role) and per-object access review. RBAC allows system administrators to check which users have access to a certain object or what kind of permissions and to which objects a certain user has access to.

Another distinguishing characteristic of RBAC is that it is policy neutral, which means that RBAC is not based on any particular policy but merely provides a means for
expressing arbitrary security policies. So with RBAC, changes in an organization’s security policy can be easily supported without remodeling its security system.

Although RBAC is policy neutral, it directly supports three well-known security principles: least privilege, separation of duties, and data abstraction. Since RBAC can be configured so that only those permissions necessary for members of the role to fulfill their job are assigned to the role, the least privilege principle is supported. Separation of duties is achieved through mutually exclusive roles, which are required when a person cannot be active in two roles at the same time if there exists some conflict in access privileges between the two roles. For example, a student cannot be a TA for the course that he is taking at the same time. Access permissions in RBAC can be as explicit as basic read/write/modify/execute operations or as abstract as any specifically defined operations on some objects, so data abstraction is also supported by RBAC.

Furthermore, a properly administered RBAC system can provide greater flexibility and breadth of application. Through the establishment and definition of roles, role hierarchies, relationships, and constraints, system administrators can control access at a level of abstraction that reflects the way business is conducted in their organization. RBAC grant users membership into roles based on their competencies and responsibilities. New memberships are established based on job assignments and user membership into roles can be easily revoked. In a RBAC system, operations are associated with roles and users are granted permission to perform operations through the roles that they are assigned to. This basic concept has the advantage of simplifying the understanding and management of privileges; i.e., roles can be updated without having to directly update the privileges for every user on an individual basis [Bark97b]. It is also
shown that the concept of an RBAC operation nicely matches with the concept of method in object technology. This association leads to approaches where object technology can be used in applications and operating systems to implement an RBAC operation.

Based on current research and experience, RBAC appears to fit well into the widely varying security policies of industry and government organizations.

1.2 What Is the Best Environment for Using RBAC?

As we have mentioned earlier, the primarily purpose of RBAC is for security management in large-scale enterprises and organizations. But with its promising advantages, RBAC deserves a thorough investigation for applications in any multi-user system with the following characteristics [Fer95]:

- Large number of users and frequent change in job responsibilities and/or functionalities.
- Only a small number of security administrators are provided.
- Large number of protected objects to be shared among users based on their job functions. This feature is due to the nature of RBAC, which promotes an organization-specific access control policy based on roles.
- Protected resources are owned by organization, not users. Access control policy is of individual organization and enforced by its security administrators.

1.3 RBAC Terms and Concepts [San96]

A Role is a semantic construct around which access control policy is formulated. The particular collection of users and permissions brought together by a role is transitory. The role is more stable because an organization’s activities or functions usually do not change frequently [San96]. It is important to note that the concept and scope of roles in
general is much broader than the concept of role in RBAC. Understanding this can result in a better design of roles for a RBAC system.

Following are the main concepts associated with RBAC:

*An object* is any resource in a system.

*A transaction* performs operations on objects. Note that all transactions in RBAC are associated with roles, but not subjects. Each role is limited to a set of transactions that a subject can perform while acting in that role.

*Access* is a specific type of interaction between a subject and an object that results in the flow of information from one to the other.

*Access control* is the process of limiting access to the resources of a system only to authorized programs, processes, or other systems (in a network).

*Administrative role* is a role that includes permission to modify the set of users, roles, or permissions, or to modify the user assignment or permission assignment relations.

*Constraint* is a relationship between or among roles.

*Group* is a set of users.

*Object* is a passive entity that contains or receives information.

*Permissions* is a description of the type of authorized interactions a subject can have with an object.

*Permission inheritance in role hierarchy* describes what conditions are required to use a role in that hierarchy as part of a transaction. There are three types of permission inheritance: *standard*, *strict*, and *lenient* [Moy01].
In *standard permission inheritance*, role R may be used in a transaction if at least one policy rule allows a role in the entry set of R to participate in the transaction, and no policy rule disallows a role in its entry set from participating in the transaction.

In *strict permission inheritance*, role R may be used in a transaction if and only if there exists at least one policy rule that allows each role in its entry set to participate in the transaction, and no policy rule prohibits any role in its entry set from participating in the transaction. Strict permission inheritance is suitable for object role hierarchies.

*Lenient permission inheritance* is the opposite of strict permission inheritance: role R may be used in a transaction as long as there exists one role in its entry set for which a policy rule allows participation, and there are no policy rules that disallow participation in the transaction for that same role.

*Resource* is anything used or consumed while performing a function. The categories of resources are time, information, objects, or processors.

*Role hierarchy* is a partial order relationship established among roles.

*Session* is a mapping between a user and an activated subset of the set of roles to which the user is assigned.

*Subject* is an active entity, generally in the form of a person, process, or device, that causes information to flow among objects or changes the system state.

*System administrator* is the individual, who establishes the system security policies, performs the administrative role, and reviews the system audit trail.

*User* is any person who interacts directly with a computer system.

*Least privilege* guarantees that only those permissions required for the tasks performed by the user in the role are assigned to the role.
Separation of duties is the invocation of mutually exclusive roles required to complete a sensitive task.

Data abstraction, such as credit and debit for an account object, can be established instead of the read/write/execution permissions usually provided by an operating system.

1.4 Roles and Related Concepts

1.4.1 Roles vs. Groups

Groups are typically treated as a collection of users and not as a collection of permissions while a role is both a collection of users on one side and a collection of permissions on the other.

1.4.2 Roles vs. Compartments

Compartments are parts of the security label structure used in the classified defense and government sectors and are based on the “need to know” semantics. Compartments are used for the specific policy of one-direction information flow in a lattice of labels, whereas roles are not confined to any single policy.
CHAPTER 2
KNOWN APPROACHES TO DESIGN AND IMPLEMENTATION OF RBAC

The above-mentioned advantages of RBAC justify its attractiveness and the motivation to apply RBAC for security management. The question is how to build a RBAC model that can best enforce access control policies. Different approaches to design and implementation of RBAC have been studied and proposed. In this chapter we address the main issues from different known approaches to design and implementation of RBAC.

2.1 A Language for RBAC Systems

Languages have long been recognized in computing as an ideal vehicle for dealing with expression and structuring of complex and dynamic relationship. A well-designed language will deliver a superior degree of flexibility in comparison to other approaches. Efficiency can further be improved by using a special purpose language, which allows for optimizations and domain specific structure. A language would make it possible for a RBAC model to express the relationships between permissions and roles in a natural way that would greatly simplify the access control system and the management of authorization policies. The main design issue of a language for role based access control system is the structuring of users, roles, and permissions. The notion of roles and other domain specific concepts are to be designed as primitive constructs within the language. The language should provide means to specify a user (a name and user identity) and the roles for which the user is authorized. To design roles, it
must be possible within a role to specify a set of permissions and RBAC need to include the concepts of role hierarchy and constraints. Since roles are intended to reflect the real world job functions within an organization, the set of attributes that the language provides would be physical, classification, time, or object attributes. Some of open issues in the design and specifications of RBAC are [Hit00]:

- How to express policies in real world system.
- How to deal with privileges, which reflect the operations of an application. This is important in choosing the level of granularity of access control, especially in object-oriented design.
- How to maintain the history of actions, which is necessary for access policies such as separation of duties.
- How to represent users and assigned roles.

This model allows access control to be independent of the rest of the system, while still able to allow or deny access. But this approach is still far from being widely accepted due to its complexity and the lack of the language’s general structure. There have been some attempts to construct access control languages, the known languages to the author are as follows:

1. **Role Definition Language** (RDL) [Hay98] allows roles to be entered by services based on credential supplied by the requesting user and the RDL program specified by a service to control access to its resources. While traditional RBAC models tend to rely on a single, per organization policy to define who undertakes each role, and what the relationships between roles are, roles in this approach are considered as more widely applicable and capable of being defined independently by services of multiple administrative domains. A
novel aspect of this schema is that it supports application specific roles and captures the relationship between the roles defined by and used within different contexts. In this model, roles can be delegated or used as credentials for authorization purposes, for example a membership in one role is a condition to enter another role. Another innovation in this model is that it supports event driven evaluation, i.e. role issuing, delegation/revocation of access rights are triggered by events (events are defined to be those occurrences that lead to a change in the state of some credential record (CR), which represents revocable delegation, group membership, or other credential that may change.) When a credential changes, its CR is modified and this information is propagated, so that all dependent CRs issued by the service are also modified. This can considerably reduce the amount of network traffic and also the access latency of authorization checks. RDL provides flexible and unambiguous specification of access control policy. Each RDL statement defines a set of conditions necessary for a client to enter a role. The syntax of RDL is defined in figure 2.

*Role Declaration*

```
def Rolename(arg, . . .) arg : type [arg : type ...]
```

*Type import*

```
import ServerName.typename
```

*Role Entry Statements*

```
Role Name([arg; . . .]) <- Role Reference [^ Role Reference . . .] [: Constraint]
Role Name([arg; . . .]) <- Role Reference [^ Role Reference . . .]
    Delegated by Delegator's Role Reference [: Constraint]
```

*Figure 2: Grammar for RDL [Hay98]*

The conditions for entry to a role are described in terms of the credentials that establish a client's suitability, together with constraints on the parameters of those credentials. For example, a health care provider may enter the role FamilyPractitioner if he provides
credentials proving that he is logged in as user John on a trusted machine. This may be written in RDL as

\[ \text{FamilyPractitioner}() \leftarrow \text{Login.LoggedOn}(\text{John}, m) \]

: \( m \) in TrustedServers.

This is equivalent to the axiom: \( X \) requests FamilyPractitioner based on certificate \( C \). \( C \) was issued by the Login service and is still valid. \( C \) is of the form “\( X \) has \( \text{LoggedOn}(\text{John}, m) \),” \( m \) in TrustedServers, \( X \) may be issued with a certificate of the form “\( X \) has FamilyPractitioner.”

2. **RSL99-RCL2000 language** is a language for specifying role-based constraints. Sandhu and Ahn [Ahn99, Ahn00a] proposed an intuitive formal language called RSL99, which uses system functions and sets as basic elements to specify separation of duty and conflict of interest policies in role-based systems. The basic elements and system functions of the proposed language are constructed from RBAC96 – a general RBAC model [San96], developed by Sandhu in 1996. In RSL99 the concept of conflicting permissions defines conflict in terms of permissions rather than roles. (Since conflict defined in terms of roles allows conflicting permissions to be assigned to the same role by error but conflict defined in terms of permissions eliminates this possibility.) To resolve possible conflicting permissions, all conflicting permissions sets and all conflicting users sets are predefined based on security policy. The formal syntax of RSL99 is constructed by using Backus Normal Form (BNF) and the semantics is described based on first order predicate logic. An algorithm called “reduction algorithm” is developed to translate a RSL expression into an expression in a restricted form of first
order predicate logic. One of the main advantages of RSL99 is the ability to express various separation-of-duty properties.

In 2000 Gail-Joon Ahn further developed the RSL99 language and proposed a language called RCL2000 [Ahn00a] as a constraint specification facility for role-based systems. In his work Gail-Joon Ahn gave a direction in how to express constraints in RBAC systems and more importantly in how to deal with those constraints.

3. Tower [Hit00] is a language-based approach to the specification of authorization policies. The most important structures in Tower are the definitions of users, roles, permissions, and privileges. Tower makes a distinction in defining permission and privilege (permission and privilege are often used interchangeably in most access control specifications). In Tower permission encapsulates the access to objects of a single class. Permission consists of a specification of the objects to which it gives access and how those objects can be accessed. The later is defined as a set of privileges. Privileges are solely concerned with methods and the conditions under which they can be accessed. Tower is a flexible access control language, which is able to specify several access control policies such as role-based access control; role hierarchy, permission inheritance, static and dynamic separation of duties, and delegation of access privileges. The drawback of Tower lies in additional implementation issues that arise when providing RBAC management in distributed environment.

4. Trust Policy Language (TPL) [Her00] aims at mapping entities to roles using well-defined logical rules. TPL allows role activation to be based on certificates that are available to the requestor. The language is defined using XML where roles are defined at the top level and under each role there are rules for role membership. The policy
language allows the system administrator to define flexible rules based on attributes in X509 certificates. However the TPL does not support inheritance, a trusted establishment system is developed to extend existing RBAC systems by mapping unknown users to roles.

5. The FAM/CAM Language [Jaj97] provides support for both positive and negative access rights. (Positive access rights specify which subjects have access to a specific object, while negative rights specify which subjects do not have access to a specific object.) The language separates access policy from access mechanism by providing the policy designer with a language that is capable of expressing any access policy.

2.2 Generalized Role-Based Access Control (GRBAC)

To extend the power of traditional RBAC, M. Moyer and M. Ahamad [Moy01] introduced generalized role-based access control (GRBAC), which incorporates subject roles, object roles, and environment roles into access control decisions. The main task of GRBAC is to make transaction authorization decision. Further more GRBAC also supports sanity checks on a policy, i.e. to detect any error in the policy specification.

Subject roles are like traditional RBAC roles. They abstract the security-relevant characteristics of subjects into categories that can be used to define a security policy.

Object roles abstract the various properties of objects, such as object type (executable, text, html, . . .) and sensitivity level (top secret, secret, confidential, unclassified) into categories. Their purpose is to classify the objects in a system based on those objects’ security-relevant properties, so that the system can mediate access to the objects based on these properties. The classification of objects into relevant object roles
can be done manually (which is a tiresome and error-prone task) or have the system automatically compute an object’s authorized role set.

*Environment roles* capture the environment information such as location, time . . . so that it can be used to mediate access control. Environment roles can also capture other state information from the system such as CPU load, network load, bandwidth, latency, etc.

Together these three types of roles offer flexibility and expressive power as well as usability of this access control model. The proposed GRBAC model is a transaction-based access control model, which means access control decisions are computed using roles and transactions. GRBAC uses role hierarchies and policy rules to decide whether to allow or deny a transaction. The object role hierarchy uses strict permission inheritance, which states that role $R$ may be used in a transaction if and only if there exists at least one policy rule that allows each role in its entry set to participate in the transaction, and no policy rules prohibit any role in its entry set from participating in the transaction. Any transaction in GRBAC can be represented using a 4-tuple of the form $T = <S, O, E, op>$, which means that a request by a user in subject role “$S$” to perform operation “$op$” on a resource in object role “$O$,” under conditions in which environment role “$E$” can be entered. The access control decisions for a transaction are given based on the following algorithm:

1. Search the policy for all rules that include all transactions $T'$ that match transaction $T$, i.e. $T' = <S',O',E',op>$, where $S'$, $O'$, and $E'$ belongs to the entry sets of $S$, $O$, and $E$ respectively.
2. If among the matchings, there is none with a positive permission bit, then deny the request by default. If there is at least one matching T’ with a positive permission bit and there is at least one matching T’’ with a negative permission bit, then deny the request by the presence of a policy conflict. Otherwise, if all matchings have positive permission bit, then allow the request.

This algorithm is easy to specify and understand but it is computationally expensive. A possible improvement to reduce the implementation cost lies in database query optimization and efficient indexing structures for the role hierarchy and policy rule database, which would make the GRBAC model fast enough to be feasible in real time practice.

### 2.3 Administrative Models for RBAC

Even though RBAC models can reduce dramatically the number of users to be controlled, in large systems the number of roles can be hundreds and the number of users can be tens of thousands. The management of roles and the relationships between users and roles can be very time consuming and tedious. Sandhu [San97] and his colleagues had the idea of using RBAC concepts for administration of roles. They introduced Administrative RBAC97 (ARBAC97) model, which uses RBAC itself to facilitate decentralized administration of RBAC. The main administrative models are administrative model for user-role assignment, administrative model for permission-role assignment and administrative model for role-role assignment. And later on Q. Munawer [Mun00] has improved these models by adding the formulation of role-role assignment and the concepts of mobile and immobile users and permissions. These models can serve as the basic administrative models for administration of RBAC.
2.4 An Object-Based Implementation of Role Based Access Control

Since the main entities in access authorization are active subjects, who try to get access to some protected passive objects, it is natural to use object technology to implement RBAC systems. It is also proved that for all object-based access control models (OBBAC), there exists a function that maps the OBBAC model to a RBAC model [Kab99]. The operations in RBAC can be implemented as methods in any object oriented programming language, and the concept of roles can be nicely matched with the concept of classes where each class corresponds to one role and consists of a set of methods/operations that a user in that role can perform. The power of RBAC mechanism is that an operation in RBAC may theoretically be anything, ranking from very simple operations (read, write, delete, . . .) to arbitrarily complex operations (a bank transaction, a medical billing process, . . .). This powerful advantage of RBAC can be perfectly exploited by using object technology, which allows operations associated with roles to be as general as possible without losing the system’s flexibility. Barkley [Bar95] proposed an object-based approach to implement RBAC, which provides flexible administration and a capability for defining complex role operations. The author proposed a three-layered structure between the secured information and applications which is

- Basic access methods are predefined in a basic access methods class, which consists of a complete set of operations based on access methods associated with the information storage mechanism. These operations are made available to an application.
• **Role classes** provide access control for the basic access methods class, one for each defined role. The methods of the role classes have the same names, types, and parameters as the methods of the basic access methods class. The role classes are in charge of access control to the information accessed by the basic access methods. The bodies of the methods in the role classes are restricted to conditionals, which determine access for the role associated with that role class and/or filters which constrict the flow of information between the application interface and the basic access methods. If access is permitted for a role, the methods of the role class then invoke the corresponding methods of the basic access methods class.

• **Application interface** includes a set of methods, which also have the same names, types and parameters as the methods of the *basic access methods* class. The methods of the application interface class invoke the corresponding methods of the role classes. Given the current role associated with the application, the methods of the application interface object select the appropriate role object.

This approach has several advantages. One advantage is that applications need not change when access conditions for roles are changed. Applications use the methods of the application interface class whose methods have the same names, types, and parameters as the methods in the basic access methods class. The methods of the application interface class and the methods of the basic access methods class are fixed and remain constant over time. Another advantage of this approach is that access conditions for roles are easily changed. Access conditions for roles are located exclusively within the role
classes. Consequently, role policy changes do not require modifications to the applications themselves.

![Diagram depicting RBAC with layered objects](image)

**Figure 3: Implementing RBAC with layered objects [Bark95]**

An important question in using object technology to implement RBAC is how to determine which principal information (user information, credential) to use for authentication of methods invocations. A possible answer to this question is due to Thomas Riechmann and Jurgen Kleinoder [Rie98]. They proposed a security paradigm called role-based principals, where object references to other application parts are associated with different roles. The authors provide a mechanism to configure principal information for method invocations on a per-reference basis. (The selection of a principal is based on object references and depends on from where the reference was obtained.)

### 2.5 Conclusion

In this chapter we have studied main known approaches to design and implementation of RBAC in different environments. Although all of the approaches try to
realize the advantages of RBAC, it can be seen that the design and implementation of
RBAC is not as simple as the concepts themselves. And sometimes it is more suitable and
profitable to build a separate RBAC system on top of an existing and well-developed
system. The idea of developing a language for RBAC seems to be very promising, but a
complete and general language for RBAC remains to be created.
CHAPTER 3
EXISTING RBAC SYSTEMS

In this chapter we study RBAC from different angles through existing RBAC systems. The idea of using the concept of roles in access control is not new. Back in the 1970s and 1980s roles were deployed in several access control products for security administration such as Computer Associates' CA-ACF2, IBM's RACF and CA-TOP SECRET. Role-based access control (RBAC) has been proposed as an alternative and supplement to traditional Discretionary Access Control (DAC) and Mandatory Access Control (MAC). Since the main components of a RBAC model are users, roles, operations, and sessions [San98] which can widely vary from application to application, RBAC should be approached as a multi-dimensional concept [San94b], where some dimension is present in a particular system but not in others. And even within a dimension the degree of sophistication of features provided by a model might also vary. Traditionally roles were used for security purposes at the system’s lower level. In 1992 Ferraiolo and Kuhn [Fer92] proposed the use of roles at application level for controlling access to application data. This innovative approach plays an important role in the development of new RBAC models, which could provide the highest flexibility and still do not loose their expressiveness. Instead of scattering security in application code, RBAC will consolidate security in a unified service which can be better managed while providing the customization required by individual applications [San94b]. RBAC can be
used separately or have other access control model building on top of it to provide 
security for more sophisticated applications with specific access control requirements. 

Even though currently not many off-the-shelf systems that implement RBAC are 
commercially available, a great effort has been done to develop RBAC models for 
different applications in various government sectors (mostly civil/federal government 
organizations), health care, business, database management, commercial enterprises, and Web servers. There is some ongoing research on developing RBAC models for 
educational applications as well.

3.1 RBAC in Business and Commercial Environment

The initial motivation for developing RBAC models was to improve the 
security management in the commercial sector. RBAC is either directly implemented or 
supported in some form in several commercially available products. More than any 
other commercial application software, DBMSs (database management systems) 
provide access control at several levels of granularity including provision for content- 
dependent [San94a] or workflow based controls. Since an application system developed 
using a DBMS can contain a large amount of data with highly differentiated access 
permissions for different users depending upon their function or role within the 
organization, database management can perfectly make a good use of RBAC 
mechanism for management of authorizations or privileges. Three popular commercial 
database management systems: Informix Online Dynamic Server (Version 7.2), Oracle 
Enterprise Server (Version 8.0), and Sybase Adaptive Server Release (11.5) have 
implemented RBAC for access control purposes. The RBAC features that are supported 
can be categorized into three main areas as follows [Ram98]:
• User role assignment,
• Support for role relationships and constraints, and
• Assignable privileges.

*User role assignment* is implemented slightly different in the three products but all of them support many-to-many relationship between users and roles and role inheritance. In Informix Online Dynamic Server a user can activate only one role at a time and have only one role active at any given time. Sybase allows users to activate multiple roles in a user session from the set of roles that have been assigned for that user. Also in Sybase a user who has been granted a user-defined role is not allowed to propagate that role to other users. This results in stronger control over role assignments and proliferation. In Oracle the user has the following two variations in role activation:

• A user can activate all authorized roles with an option to selectively exclude some authorized roles from getting activated for the session.

• A user can deactivate or disable all the roles for the current session.

In Sybase, the user-role assignment is a centralized administrative task, where only the System Security Officer can assign roles to users. In Informix and Oracle, it is possible to implement this as a discretionary access control mechanism by enabling the role grantee to grant that role to other users. While Sybase and Oracle allow multiple roles to be activated in a user session, Informix has only one role can be active at anytime.

*Support for role relationships and constraints* is a powerful feature of RBAC, which is the ability to define mutual exclusion of roles. There are two types of mutual exclusion that have been defined:
• Two roles are in static exclusion if a user cannot be granted both roles and
• Two roles are in dynamic exclusion if a user cannot activate or enable both roles at the same time.

In this way Sybase RBAC implementation provides for enforcement of both static and dynamic separation of duty policies. Oracle and Informix do not directly support separation of duties policies and all of the three products do not implement cardinality constraint (the ability to restrict the maximum or minimum number of users that can be authorized for a role).

Assignable privileges is a way to level different access control policies into sets of access permissions. Privileges can be grouped into three categories: database-level privileges, table-level privileges, and execute privileges (Informix). In Sybase privileges are categorized as object access permissions and object creation permissions. Oracle privileges are categorized as system privileges and object privileges. In Informix only DBMS Object-level privileges can be granted to the roles while in Sybase and Oracle both System-level and Object-level privileges can be granted to roles.

RBAC’s distinguished features also support its smooth adaptation to knowledge management and data warehousing systems. Megaache, Karran, and Riberio Justo [MKJ00] have defined a RBAC model for CODA-based (Complex Organic Distributed Architecture) business intelligence enterprise systems, and derived a CORBA-based architecture for this model, which supports scalability, effectiveness in ensuring secure access, and good performance. They proposed a security system architecture which is characterized by the symmetry of the security components in different layers on one hand and the increase of the administration levels between those layers on the other hand.
CODA supports some of the main features of RBAC policies. Firstly, the architecture provides role hierarchy, which is separated into role usage hierarchy and role activation hierarchy. One key feature of CODA is that activities in the system are divided based on their complexity and type into layers. Tasks at each level are closely associated with roles (role activation hierarchy). Users of higher layers may acquire access to detailed data belonging to the layers below (role usage hierarchy). Secondly the separation of duties constraint is expressed in term of mutually exclusive roles. CODA is a cybernetic-based model using the VSM (Viable System Model) architecture, which provides a natural way to separate conflicting roles between the different layers of the system. Other constraints are supported when assigning permissions to roles.

The security administration is done via User-Role Assignment (URA), permission role assignment (PRA), and security administrator, who controls and monitors the URA and PRA servers in addition to the access, authentication, and audit servers. The Security Administrator is responsible for the formulation of the security policies and to propagate them to the other servers. It is also responsible for adding and deleting users and components from its layer. In general the model is very efficient in modeling evolving behavior, which is a key requirement in the development of enterprise knowledge management systems.

3.2 RBAC for Network Environment

Looking at the current available RBAC products we have noticed that RBAC shines the most in the network environment, where the management of a large number of subjects, objects and their associations can be greatly simplified by using RBAC mechanism. Network communications is one of the main discoveries in this century,
which open a new era in information technology but also at the same time never than before information security becomes such an urgent and important problem. Communicating in an open network means open your door to everybody unless you have some way to guard the door. People are so different, so are the problems we have to deal with in this kind of communication. This explains the large number of RBAC products offered for network security as well as the diversity in the complexity of these products which ranges from pretty simple RBAC system for intranet security [Tar97] to a very complicated RBAC systems for web servers [Bark98], [Shi01], Internet [Fer98], Network enterprises [Tho98], . . . which include all or most of the advanced components of RBAC. Section 3.2 will walk us through the most important functions provided by the above RBAC products for network environments.

3.2.1 RBAC for Intranet Security

The main functions of a RBAC system for intranet security developed by Tari and Chan [Tar97] are to determine:

- if a network object has appropriate permissions to access intranet resources, and
- how to check the permissions of a network object.

The system is implemented by using active network objects (agents), which contain an interface (that specifies the required security procedures to be enforced within an Intranet) and an implementation part (which describes how the security procedures can be performed within an intranet’s server to support the specified agent’s interface). The access control information is stored in the system’s RBAC database, which consists of:
• a local role database which stores the access information of a server’s network objects within an intranet and different numbers of permission domain tables to express all accessible network objects that a role can access and
• a global role database which records all accessible network objects from any intranet server and stores all the authorizations for the corresponding global roles. (global roles are “ordinary enterprise roles that have global permissions on a network server”)

3.2.2 RBAC for Network Enterprises

The main purpose of the RBAC for network enterprises system [Tho98] is to support a fine-grained access control policies without losing the ease of management. The system is implemented using divide and conquer method, which separates the security tasks into two groups, one group is to be performed by application developers and the other is the responsibility of local system administrator. The system consists of seven layers which allow an administrator to “quickly implement a course grained security policy, or precisely specify a fine grained policy by further customizing the policy using the more detailed layers of abstraction.” The seven layers provide the following functions:

Layer 1 – Objects: organize the system objects into applications, to which they belong.
Layer 2 – Object handles: specify how the objects are to be used and how the access to objects can be provided. For example, instead of assigning each method to a role individually, this role’s access permissions to an object (methods) can be grouped into a set before being granted to that role.
Layer 3 – Application constraints: can be time-based, history-based, attribute-based constraints that specify which individuals have access to specific object instances.

Layer 4 – Application keys: is a collection of security information encoded by the application developer for use of local system administrator.

Layer 5 – Enterprise keys: are used by local system administrator to assign access to users. There is a one-to-one mapping between enterprise keys and application keys.

Layer 6 – Key chains: is a collection of enterprise keys used by local system administrators to build up a complex role hierarchy for their site.

Layer 7 – Enterprise constraints: are the constraints that can span several applications. These constraints are applied to key chains only.

<table>
<thead>
<tr>
<th>7. Enterprise constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local System Administrator</td>
</tr>
</tbody>
</table>

<p>| 6. Key chains |
| 5. Enterprise keys |</p>
<table>
<thead>
<tr>
<th>4. Application keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application developer</td>
</tr>
<tr>
<td>2. Object handles</td>
</tr>
<tr>
<td>1. Objects</td>
</tr>
</tbody>
</table>

Figure 4: A layered RBAC structure

3.2.3 RBAC for Different Types of Internet Documents

Since nowadays network communication involves different types of information (audio, video, animation, graphics, and text) and information transfer protocols, the type of documents being transferred through the Internet becomes a considerable factor in access control design. In fact some of the recent RBAC models are developed for specific type of Internet document, namely RBAC model for XML repositories [He00], RBAC
model for digital content [Wei01], and RBAC for hypertext/multimedia document [Fer98]. RBAC model for XML repositories (RBXAC) consists of four components:

1. **XML database.** The XML database is the central component of the model. It is responsible for loading and storing XML files. It stores both access control file (in XML format) and XML files. The access control file includes a list of users, an operation tree, and a role tree.

2. **Access control layer (ACL).** The ACL is a thin layer wrapped around the XML database and can be regarded as an application of the XML database. It intercepts all requests from users and prepares answers for each user session by querying the XML database and check permissions according to the role of the user.

3. **User session.** Each user session receives inputs from a user and sends requests to the ACL on behalf of the user. It also interoperates answers from the ACL to the user.

4. **Administration session.** The administration session interact directly with the database so it can modify the access control file directly. Its responsibilities include granting/removing XML node access permission to roles, creating new users (introducing new users into the system), deleting old users (this is allowed only when the users have no role memberships), assigning/depriving roles, creating new roles, and deleting old roles.

The RBXAC model requires that

1. A role may control more than one role;

2. A role may create a new role subject to two conditions: A) the role is allowed to create a new role; B) the new role is a role that the creator controls;
3. When a new role is created, it has no control over any other roles;

4. When a new role is created, its creator has no more controls on it than on its parent; and

5. a role can be given no more privilege than it is necessary to perform its job (this requirement is based on the least privilege principle).

This role-based access control model for XML information management supports both discretionary and mandatory access control, precise control over the data, separation of duty, and lightens the administration tasks.

**RBAC for digital content:** RBAC is combined with public key cryptography to protect digital content, which is encapsulated into secure document containers. Digital content can only access via a secure container. This document container checks access rights as specified in a RBAC model and encryption enforces these control mechanism.

**RBAC for hypertext document:** is a three-layer system: View/Presentation layer, Model layer, and Storage layer. The user request is obtained from the view/presentation layer (which can be represented using object-oriented web server) and sent to the model layer. The model layer (which can include a complete enterprise model) includes authorization rules together with the abstract description of the documents. The authorized portions of the documents are assembled by the model layer using the mappings to the storage layer (which can be a relational system, an object-oriented DBMS or a combination) and returned to the user.

### 3.2.4. RBAC for WWW Servers

RBAC provides a rich model for managing information and security since many other security models can be represented as subsets or simplifications of one of the four
RBAC control models [San96b]. Furthermore RBAC helps to reduce the management cost and the number of errors occurred when managing a large number of users and network resources in distributed multimedia environments, such as World Wide Web (WWW) [San96b]. Based on these motivations, Barkley, Cincotta, and Gavrilla [Bark98] at NIST have implemented RBAC on the web (WWW/Web) for both UNIX (e.g., for Netscape, NCSA, CERN, or Apache servers) and Windows NT environments.

The primary purpose of RBAC/Web is to provide a flexible and effective access control for intranet computing environment and it can be incorporated into existing systems with no modification to server code, hence make it portable to virtually any web servers. RBAC/Web implements many of the RBAC components, such as role hierarchy, permission/role inheritance, separation of duties, and role creation/revocation constraints.

The RBAC/Web components are listed in Table 1.

<table>
<thead>
<tr>
<th>Database Server</th>
<th>Hosts the authoritative copies of the files which define relationships between users and roles, the role hierarchy, and the constraints on user/role relationships. These files are created and maintained by the Admin Tool. When changes are made these files, the Database Server notifies the Web Servers to update their cached copies.</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Library</td>
<td>A specification which may be used by Web servers and CGIs to access the RBAC/Web Database. The API is the means by which RBAC may be added to any Web server implementation. The API Library is a C and Perl library which implements the RBAC/Web API.</td>
</tr>
<tr>
<td>CGI</td>
<td>Implements RBAC as a CGI for use with any currently existing Web server without having to modify the server. The RBAC/Web CGI uses the RBAC/Web API.</td>
</tr>
<tr>
<td>Session Manager</td>
<td>Manages the RBAC Session. The RBAC/Web Session Manager creates and removes a user’s current active role set (ARS).</td>
</tr>
</tbody>
</table>
The RBAC/Web implementation solves the possible conflict between inheritance development and dynamic separation of duties relationships among roles by enforcing a consistency principle: the inheritance, static separation of duty, and dynamic separation of duty relationships among roles must be consistent at all time. Also the permitted operations implemented by RBAC/Web are the “methods” defined in the HTTP protocol definition. These methods are GET, HEAD, PUT, POST. RBAC/Web controls the ability of a user acting in a role to perform an HTTP method on a URL.

### 3.3 RBAC in Health Care Environment

Within the health care industry, there are continuing problems associated with how to ensure the security and integrity of health care information, in particular, patient information. These problems will only become larger in the future with the increasing automation and integration of health care information.

In their project "Distributed Communication Methods and Role-Based Access Control for Use in Health Care Applications," Poole et al. [Poo95] studied the applicability of RBAC to health care information. Since there are several variations on the RBAC model, there is a question of which variations are most suitable for health care information. The authors demonstrated the usefulness of RBAC to health care by using RBAC with patient records. The demonstration suggests different roles that are appropriate with patient records and defines sample operations associated with those roles. They also produced a sample of RBAC policy related to clinical and administrative patient data. The policy is RBAC with the addition of the capability of labeling information that is only available to the patient and the doctor. It specifies roles and the level of access permitted by each role.
3.4 RBAC in Multi-Level Security Systems

To reduce the number of objects to be controlled, a security model is developed by the US military called multi-level security (MLS) system, where secure objects are grouped together into different levels classified as TOP SECRET, SECRET, CONFIDENTIAL, and UNCLASSIFIED. This model (MLS) has been well studied, thoroughly tested and widely used throughout military services and elsewhere. Even though MLS is a very trustable security system, implementation of the MLS "kernel" requires that each user must be separately assigned numerous "privileges," i.e., the right to access certain objects or groups of objects within the system. In a large MLS system, with hundreds or thousands of users and objects, the assignment and maintenance of the user-access privileges to various objects relation is a substantial administrative burden. Particularly when an individual changes job assignments, all connections between that individual and objects to which he/she previously had access must be revoked, and new connections must be established to all objects needed for performance of the new function.

In 2000 R. D. Kuhn implemented RBAC in MLS system and received a US patent for his invention. The main purpose of the invention was to provide a method whereby access of individuals and subjects to objects controlled by an MLS system can be simplified using RBAC as an interface to the MLS system [Kuh00] without interfering the kernel of existing MLS system. In his implementation, R. Kuhn has “injected” RBAC into a MLS system by establishing a relationship between privileges within the RBAC system and pairs of levels and compartments within the MLS system. RBAC is implemented on an MLS system by establishing a mapping between privileges within the
RBAC system and pairs of levels and sets of "compartments" assigned to objects within the MLS system. This implementation strategy realized all the advantages provided by RBAC without loss of the security provided by MLS. To implement RBAC in an MLS environment, a trusted interface function is developed to ensure that the assignment of levels and sets of compartments to users is controlled according to the RBAC rules; that is, the trusted interface ensures that the user-role assignment (URA) and permission-role assignment rules are followed rigorously. The interface also provides a proper mapping of the roles to corresponding pairs of levels and sets of compartments. Access requests from subjects are first mapped by the interface function to the pairs of levels and sets of compartments available to the corresponding role. Then access to the objects is controlled entirely by the rules of the MLS system, responsive to the pairs of levels and sets of compartments assigned. In essence, each user’s request for a privilege is checked to ensure that it is permitted to the subject's role at the time of the request. The trusted interface then sets the subject's compartments and levels according to a mapping function that determines a unique combination of compartments and levels for the privilege requested.

### 3.5 Other Role-Based Access Control Models

Besides the above-mentioned RBAC models there are other access control models that implement RBAC concepts such as a model that uses RBAC to improve concurrency in distributed object-based environment [Kim99], uses SESAME to implement RBAC in UNIX file system [Ash99], or uses a RBAC model that enables the management of access control policy for the courses in a virtual university [Jae99]. Another powerful and flexible role-based access control model in collaborative environments was developed by
Shen and Dewan [She92], where policies must account for concurrent operations on shared objects and other complex access issues.

3.6 Conclusion

In this chapter we have studied some of the existing RBAC systems that were implemented for various security purposes in different environments. The main functions and important design and implementation features of each system were discussed and analyzed. Many systems have implemented some or most of the main concepts of RBAC (such as hierarchical structure, inheritance, delegation/revocation of access privileges, separation of duties, least privileges . . .) to maximize the advantages of RBAC. One of the promising areas for RBAC is security for the Internet environment [Eps99, He00, Ahn00b, Bark97b]. There remains a lot to be developed since it is a very large and open environment. For strong acceptance to continue, RBAC must be nicely integrated into the framework of the Internet.
CHAPTER 4
EXTENDING RBAC MODELS WITH NEW CONCEPTS

The main purpose of traditional access control mechanisms is to protect passive objects from unauthorized access from active subjects. The rapid development in all areas of information technology, especially in distributed environments, has introduced new situations where traditional access controls do not cover yet. For instance, both MAC and DAC do not thoroughly consider the dynamic behavior of the system in making access control decision and they were not designed to provide access control to a large pool of widely dispersed objects of different types and subjects from heterogeneous environments. To resolve new access control requirements, new access control elements have been studied and developed. In this chapter we analyze RBAC in conjunction with new access control elements such as information flow control [Iza01], workflow control [Ahn00b, Pay99], state [Ste01], session [Gut01, Lee99], type [Hof97], and dynamic RBAC [Che99]. These elements are included neither in traditional RBAC models [San96b] nor in the recently proposed RBAC standard [Fer00]. We show how RBAC can be used to construct a powerful and effective security system to fit very different access control requirements.

4.1 Control Information Flow in Role-Based Access Control

The main purpose of any access control in an object-based system is to protect objects from being read or manipulated by unauthorized subjects. The access rules dictate which operations are allowed to performed on a protected object by a given subject. But
this access rule alone might not cover the case where an illegal access can “legally” be obtained through accessing other object(s). This is where the notion of secure information flow and information flow control takes place. The flowchart in Figure 5 demonstrates possible illegal information flow.

![Figure 5: Illegal information flow](image)

Assume subject S1 has read/write access rights to object O1 and O2 while subject S2 can only read O2.

1. S1 reads O1,
2. S1 then writes O1’s data into O2, and

The above scenario shows that even though S2 is not allowed to access O1, it can still get access to O1’s data indirectly through O2. The beauty of RBAC is that it separates direct connection between subjects and objects. Access permissions are assigned to roles and users are assigned to roles in order to get access to protected objects. But that does not eliminate illegal information flow in roles. A solution to this problem for object-based systems are proposed by Izaki, Tanaka, and Takizawa in their
paper “Information flow control in role-based model for distributed objects” [Iza01]. The main idea of this approach is to define a set of save roles where there is no illegal information flow among objects and to check where illegal information flow can occur and find a way to safely perform transactions belonging to unsafe roles. To do that four parameters are introduced to characterize each method: input type, manipulation type, derivation type, and output type. (Methods represent operations that a subject can perform on an object and they can be as simple as read/write/execute and as abstract as bank transactions). These four parameters are used to detect all possible information flow of different objects as the result of an execution of a particular method on those objects. Safe roles and safe role sets are defined based on definition of safe information flow and a directed graph $G(R)$ named flow graph. Information flow is then controlled through roles. An algorithm called Information Flow (IF) is developed to check if illegal information flow can occur after a method is performed, in which case the method is rejected. For example, using the same Figure 5, the illegal information flow can be detected using the IF algorithm. As we mentioned earlier, the information flow from object $O_1$ to $O_2$ can be illegal if subject $S_1$ (in role $r$) derive some data from $O_1$ and output it into $O_2$, after which subject $S_2$ (in role $r'$) reads and manipulates this data when accessing $O_2$ even though $S_2$ is not allowed to access $O_1$’s data. Since the information flow algorithm checks on safeness condition for every method performed by a role on an object/objects, the information derived from $O_1$ that may flow into $O_2$ as described above will cause the safeness condition of the “write” method of $S_1$ to fail. Hence $S_1$ cannot write to $O_2$. Using this IF algorithm, the illegal information flow in Figure 5 can be solved by letting $S_2$ perform the “read’ operation on $O_2$ before $S_1$ writes to $O_2$ is shown in Figure 6.
Information flow control is a very important aspect in any access control system. There is more than one way to resolve this problem in RBAC. Another possible approach is to carefully define sets of constraints that prohibit all possible illegal information flow.

Information flow exists not only in object-based systems but in other systems as well. We think that information flow is not a stand-alone concept but rather has very close connection with other concepts such as state, session, and transaction. It is important to incorporate these concepts in the access control decision-making process.

### 4.2 RBAC for Workflow Management

Along with information flow control, business process control (or workflow control) has become an important issue in many access control systems, especially in heterogeneous distributed computing systems. A workflow is defined as the computerized facilitation to automation of a business process, in whole or part [Hol95]. Workflow technology is a promising solution for protecting business assets, because it controls not only who has access to what but also when that access occurs [Pay99].
Workflow and RBAC share several fundamental concepts such as roles, methods (operations), and performers. Workflows are enforced by a workflow management system (WMS). The user interacts with the WMS to gain access to resources controlled by the workflow.

Abbott and Sarin [Abb94], Feinstein et al. [Fei95], and Bertino et al. [Ber97] note that workflow management can be simplified considerably by adopting an RBAC model. For example NAPOLEON (Network Application POLicy EnvirONment), a multi-layered role-based access control modeling environment, is a good candidate for workflow policy management [Pay99] since it includes the role constraints that are required for workflow [Ber97].

![Diagram](Figure 7: A role-based perspective of workflow [Pay99])

A role-based perspective of workflow, its fundamental concepts and the relationships between them are depicted in Figure 7. A *performer* is a person who acts in
a role that provides him with a set of accesses to perform some steps in a system’s working process. A step is like a sub-role which defines a group of accesses that are specific to a task. A role is a collection of steps and their associated workflow conditions, under which these steps can be performed. A workflow condition is often expressed as entry and exit conditions on the step, that is, the step can begin when and can end when the conditions are true. A step can be performed using one of several methods. Methods use and produce work products. In other words a work product is an artifact created or modified by methods.

In recent years web-based workflow systems became very popular due to its ability to support dynamic business processes in distributed computing environments. To simplify the security management and improve security services of web-based workflow systems, Sandhu and his research team [Ahn00b] have implemented a role-based access control into an existing web-based workflow system. The authors use X.509v3 certificates with role attribute and employ a user-pull style where the client requests a client’s certificate from the role-server and presents it to the workflow system. Their major goal is to have minimal changes to the existing web server and no changes to the browser. For simplicity they defined a permission to be the authorization to execute a task in a workflow system. The authorization is enforced in terms of roles. Each task can only be executed by users belonging to a specific role. The proposed RBAC employs many-to-many user to role and also many-to-many permission to role assignment relations. The main components of the system architecture are workflow design tool, role server, and web-based workflow system.
The workflow design tool is responsible for specifying information flow and dependencies among tasks, creating roles and role-hierarchies, and assigning a role to each task.

The role server has two major components: user-role assignment and certificate server. The user-role assignment component maintains role hierarchies, assigns users to roles, and generates and maintains the user-role assignment database. Certificate server authenticates clients, retrieves client's role information from user-role assignment database, and issues certificates with client's role information.

The workflow system consists of web-based task servers, which enforce authorization in terms of roles so that a user can execute a task only if the user holds the appropriate role. Authorization of users for tasks (by means of roles) is verified by using Java-based and SSL-enabled web-servers.

Another implementation of RBAC for workflow management is done by Barkley, who develops a method for employment of RBAC techniques for controlling the ability of individuals to carry out operations within a workflow process. The innovation in this project ("Workflow management employing role-based access control") [Bark00] is the deployment of roles from different aspects. Besides standard use of roles for access control [San96b], a role sometimes is used to refer to the set of operations to which the permission(s) associated with that role grants access.

RBAC is not only a powerful tool for workflow management policy as shown above; it can also be used as a means of implementing workflow. The argument is since
access to a resource can be enforced by means of roles, assignment of a permission to perform an operation and the removal of such an assignment can be used to sequence a set of operations. The sequencing of operations is the fundamental behavior required to support workflow.

4.3 Extending RBAC Model with States

One of the main advantages of RBAC that make it a powerful means of describing access control policy is its policy neutral feature. Any security policy can easily be described using RBAC components. However, the notion of state has never been included in RBAC until recently, when Steimuller and Safarik [Ste01] finally introduced this concept into RBAC. Traditionally, changes to the components of a RBAC model such as adding or deleting a role, assigning new permissions to a role, . . . are viewed as a transition to new security policy. Introducing the notion of state and state transition into RBAC models enables us to view the changes of components inside a RBAC system as transitions between states of an access control policy and to review properties of the policy, the dynamic behavior of a policy as well as to compare the policies. Based on the fact that access control policy defines permissions granted to a particular user, the authors conclude that only user assignment (UA) and permission assignment (PA) relations determine access control policy. Since users, roles, and permissions not used in UA or PA have no impact on access control policy, the only changes of RBAC components that affect access control policy are the insertions and removals of elements to and from UA and PA. “Elementary state changes” are defined as insert/remove an element to/from UA or PA. The graph of state transitions for a given set
of users (U), permissions (P), and roles (R) can be constructed based on a set of partial ordering on the set

\[ \text{CONF} = (U \times R) \times (P \times R) \]

of all possible configurations. Every node of a transition graph represents the state of the access control policy, i.e. one configuration of the RBAC model components. Every state is associated with a particular user-permission (UP) relation. Let \( U = \{u_1, u_2\} \), \( R = \{r\} \), \( P = \{p_1, p_2\} \), Figure 8 is an example of a state transition graph of these given sets.

![Figure 8: An example of state transitions graph](image)

Note that multiple states can be associated with the same user-permission relation. The security policy becomes dynamic with all of its possible states and transitions between them. Role hierarchy has no impact on the state transitions since it only modifies the user-permission relation in comparison with the basic RBAC model, where there is no role hierarchy. Constraints are just various predicates upon components of RBAC model, which determine allowed configurations of RBAC components for a particular policy, so the state transition graph of a RBAC policy can be considered as a subgraph of equivalent RABC policy without the constraints.
Extending RBAC model with states is a new idea that worth more investigation into it. There are a lot of open issues to be further developed such as the types of changes of states, the relationship between states and user permission relation and how to visualize the dynamic behavior of the system.

### 4.4 RBAC and Session Management

Unlike workflow and information flow, the notion of session is included in RBAC as a means to support separation of duty (SOD) and least privilege properties, especially dynamic separation of duty, where user can be assigned roles-in-conflict as long as they are not activated in the same session. Traditionally session is defined as a logical duration of interaction between system and user from user login to user log out. But in an automated information processing environment, session is often referred to as a dynamic process, during which user performs a particular set of tasks. In RBAC session is a mapping between a user and an activated subset of the set of roles to which the user is assigned, which means that session can be dynamically invoked and assigned to a user within the security policy.

One of the attempts to explore the features and properties of session is discussed in Lee and Noh [Lee99], where session is used as a vehicle for building applications. The idea is to use the concept of session in building a session-based integrity enforcement application. For this purpose three new relations are introduced, which are mutually exclusive sessions, session-user assignment (SUA) and session-role assignment (SRA). Since the permission-role relationship (i.e., the assignment of permission to role) is stable in comparison to the user-role relationship (i.e., the assignment of user to role), SRA can be performed in application design phase and SUA in application operation phase.
There is a trade off between system performance and flexibility which depends on whether user and roles are assigned to a specific session during application design phase or whether user and roles are assigned to a session during operation phase (i.e., during runtime). In the first case the system performance may be enhanced (i.e., less overhead), but the flexibility might be diminished (since users and roles are tied to a specific session at design phase). In the later case the system flexibility is increased but the performance can be hindered because of possible overhead in managing the dynamic SUA and SRA relations. The main contribution of this study is the idea of using session as a dynamic component in building integrity enforcement application. Deploying this extended and dynamic view of session can also add another advantage of RBAC model as a powerful access control mechanism for workflow in distributed systems.

It is worth mentioning that the concept of session is not only an important concept in RBAC, in fact it is a well-known concept in many areas such as database management and network management. In some systems session management plays an important part in managing the system’s security, especially in the authorization process. In the Web environment, for example in HTTP environment, session management provides many services to system security such as time service (related to session duration and time-out require agreement on the time), user profile service (provides user attributes, particularly security roles and distinguished names) and ticket issuance service (grants a session ticket to an authenticated user). These Web session management services can be used with RBAC models for network security purposes as proposed in Gutzman[ Gut01]. This is an approach that uses RBAC and Web session management to protect against network security breaches in the HTTP environment, where the RBAC and session management
services augment network-level security, such as firewalls. The RBAC is implemented through the Internet Engineering Task Force’s Lightweight Directory Access Protocol (LDAP). Session management is implemented through cryptographically secured, cookie-based ticket mechanisms.

Even though session is an important concept in RBAC not as much research has been done on session with respect to separation of duty, role hierarchy, constraints, or RBAC administration. There are still a lot to be done to explore all the features and functionalities of this concept, which can greatly contribute to the development of access control models in general as well as in RBAC models in particular.

4.5 RBAC and Type Enforced Systems

In this section we examine the connection between RBAC and another access control mechanism, namely type enforcement. The concept of type does not exist in current RBAC models, but Hoffman [Hof97] has discovered an interesting application of RBAC concepts in the administration of a type-enforced operating system. The motivation of this approach lies in the observation that access control decisions should depend not only on the role an untrusted subject is in at the time he makes a request for a protected resource. Since type enforcement restricts the accesses a subject can have to objects through the use of domain labels on subjects and type labels on objects, adding an additional lower level mechanism such as type enforcement to RBAC mechanism produces more secure systems. The combined access control mechanism connects roles to domains, where each role is associated with a set of domains.

Figure 9 describes the access control process in a type enforcement system enforcing RBAC. The request for a resource is first checked based on RBAC policy.
Once roles are identified, along with their associated duties, a collection of subjects and domains, necessary to perform those duties, are identified. Next, type enforcement facilitates a finer layer of control, where each subject is assigned a domain (subject -> domain) and each object is assigned a type (object -> type) and a (Domain, Type) pair determines which set of permitted accesses that subjects operating in that domain can perform on objects of the specified type. It is shown that introducing RBAC into type enforcement systems facilitates system administration and system creation. Also many of the high level RBAC features can be easily implemented through appropriate administration utilities on a type enforcing system.

![Request for resources](image)

**Request for resources**

**RBAC layer:**
- Operations -> Role
- Role -> Domains

**TE layer:**
- Subject -> Domain
- Operation -> Type

**OS:** permission bits

![Access control process in a type enforcement system enforcing RBAC](image)

**4.6 Conclusion**

In this chapter we have inspected some of the new concepts that are not present or less studied in the current RBAC models such as information flow control, workflow,
state, domain, and type. We have shown that these concepts are very important and cannot be neglected in access control decision-making process. We also emphasize the need to further develop the concept of session in RBAC systems. It would be very interesting to explore the interconnection between these concepts, for example is there any relationship between information flow and system state, information flow control and role/permission inheritance? In some of the recent literature on RBAC a derivation of role definition for enterprise systems is introduced, where role can be defined as a set of tasks [Par01]. There is a good reasoning for that view since task is a meaningful unit of business work and access rights are required only for executing assigned tasks, in other words access rights can be assigned based on a user’s tasks. Also there is a strong connection between information flow, workflow, and tasks performed in the system. The problem is introducing tasks into RBAC will add a substantial complexity to the access control mechanism and diminish the primary goal of RBAC, which is the simplicity of the security management. Furthermore a task-based RBAC cannot support the delegation of access rights, which is one of the main characteristics of RBAC. Unless an automated task derivation process is provided, this would be a trade off between cost, complexity and an efficient user interface that supports real world style access control, i.e., the interrelation between change of real world and change of RBAC schema information.
The purpose of this chapter is to demonstrate how the RBAC concepts can be applied to the development of a RBAC system. As an example we show the steps in design and specifications of such a system assuming a plausible CISE department access control policy.

5.1. Role Creation

In Figure 10, we define five levels for the roles in the system policy based on the organizational structure of the department. At the root of the hierarchical structure of the roles is the CISE system user, which means that all the roles in the system policy are
derived from the role “CISE system user.” The hierarchical structure reflects the inheritance rule, such that the roles at the lower level in the role structure inherit all the access privileges from their parent nodes and above. For example, an undergraduate student inherits all the permissions of the role “student,” as well as those of role “CISE system user,” i.e., a undergraduate student’s permissions are the union of the permission sets assigned to role “student” and role “CISE system user.”

At the second level are the roles “faculty,” “staff,” “student,” “guest.” The role “staff” is further divided into “system staff” and “administrator staff.” Under “student” we define roles “undergraduate student,” “Postbac,” and “graduate student,” which is further divided into roles “PhD student” and “Master.” Role “TA” is derived from roles “PhD” and “Master” meaning a TA can be either a Master student or a PhD student. These are the main roles in the organizational structure of the department. Each role is associated with a set of different duties and responsibilities; such as a professor has teaching duties, research, and other services in different committees. To fulfill all of his/her duties, a professor will need a set of privileges, for example permission to access students records, access to computers (printing, copy, computer account, . . .) as well as access to research labs if he/she is involved in some project. Also note that in our definition of roles, we follow the organizational structure of the CISE department, so even though a professor role can be in two different roles as a teacher and a researcher, we define teaching and research as two of the functionalities of a professor. This also shows the flexibilities of RBAC, it is up to an organization to define the roles and a set of duties appended to each role based on its specific security policy.
5.2 System Components Definition

We define our system components based upon the NIST’s recently developed RBAC standard [Fer00].

5.2.1 Data Elements

*Users:* all CISE students, faculty, staff, and guests.

*Roles:* faculty, staff (system, admin), students (undergrad, postbac, grad (PhD, Master, TA)).

*Objects:* disk space, email, Internet resources, personal web page, modems, printers, labs, consultants and on line help, backups, electronic student records.

*Operations:* all possible operations on the above objects, such as read/write/execute personal data, printing, browsing the Internet, working in labs, creating and maintaining personal web page, etc.

*Permissions:* different access permissions to the above objects, such as using email and internet services, printers, having backups for personal files, etc.

5.2.2 System Requirement Specifications

*Administrative functions* include creation and maintenance of element sets such as users, roles, objects, and operations. Objects and operations are predefined by system administrator. Our system has predefined objects such as printers, computers, and labs and predefined operations such as working in the labs, using computers, printing documents, etc. Each CISE system user is assigned one or more roles based on their status and job functionalities. A set of access permissions is predefined for each role based on the principle of least privilege and separation of duty, which are supported by defining mutually exclusive roles. Only system administrator has the right to add a new
user (add a new student to the system by assigning him a CISE account) or delete a user (remove a already graduated student 6 months after his graduation), as well as create or delete roles. Also the system supports many-to-many user-role assignment and permission-role assignment, for example a user can be assigned role “PhD” and “TA” and role “student” is assigned to all users, who are students. Each role is associated with a set of permissions, for example role “guest” has a set of permissions such as using email, printing documents (with constraint on quota), and browsing the Internet.

Supporting system functions: we assume that our system user can activate one role for each session he creates. So before starting a new session the user has an option of choosing one role from his assigned role set to be the active role for that session.

Review Functions: using the system RBAC database administrator can view all the users assigned to a given role and all the roles assigned to a given user as well as the list of permissions assigned to a given role or the list of permissions assigned to a given user.

The system can support other advanced features of RBAC as follows:

- Inheritance: Our system has a hierarchical structure as shown in Figure 10 and it supports multiple inheritance, i.e. the lower level roles inherit access privileges of their parent and the roles at above levels all the way up to the root in the role hierarchy. For example, a PhD student inherits all permissions of role “student”, which in turn inherits all permissions of role “CISE user.” So the permission set of role “PhD” is the union of permissions of role “PhD,” “student,” and “CISE user.”

- Constraint: A student can have certain privileges as a TA, and the privileges will be revoked when the term, in which he is a TA terminates (time constraint). Also there
are constraints on the disk quota and printing quota. Each graduate student can be TA for only one course during any semester (cardinality constraint).

- **Separation of duties**: The system supports some features of static separation of duties and dynamic separation of duties. For example, TAs are only responsible for grading home works and/or exams but the letter grades are given only by the professor teaching that course (static separation of duties). So the TAs have no access to the letter grade of the course, in which he/she was a TA. Also TAs have access to student records of the course he is a TA for, but this privilege is revoked as soon as the semester ends (dynamic separation of duty). Furthermore, the whole staff provides services for the department, but the administrative staff’s duties are different from the system staff’s.

### 5.3 Suggested System Design

Figure 11 depicts our suggested system architecture, which consists of two components: system management and role server.

#### 5.3.1. System management

The system administrator creates the role hierarchy and assigns a set of permissions for each role based on the security policy (in our case the CISE security policy). The permission-role assignment supports permission inheritance, which is dictated by role hierarchy. For example, the permission-role assignment for role “PhD” must take into account that a user in role “PhD” has all permissions of roles “CISE user” and “student” as well as permissions, designated to role “PhD” alone.

#### 5.3.2. Role Server

The role server has two major components: user-role assignment and certificate server. The user-role assignment component maintains role hierarchies, assigns users to
roles, and generates and maintains the user-role assignment database. All roles of a user are explicitly assigned in the user-role database. In other words the role hierarchy is simulated by explicit user-role assignment. This simplifies the task of the certificate server. For example, when a student becomes a TA, his new role will be explicitly assigned. Certificate server authenticates clients, retrieves client’s role information from user-role assignment database, and issues certificates with client’s role information.

**Figure 11: Suggested system architecture**

The access control process proceeds as follows: first a user presents his username and password to the system server to be authenticated. After a successful authentication the user has the option of choosing one of his assigned roles to be the active role for his current session. The system management then sends an authorization request to role server to retrieve role information. The authorization certificate is then sent back from.
role server to system management, which in turn checks the authorization status based on its permission-role assignment information. After a successful authorization, the user can execute transactions in the server based on the user's role instead of his identity.

5.4 Conclusion

We have shown that RBAC model can greatly simplify the administration management. Any security policy can easily be specified using RBAC concepts. Since RBAC is policy neutral, it can be configured to match any system policy. The recently available RBAC standard [Fer00] not only serves as a fundamental guideline for RBAC products but it also puts a milestone in the development of RBAC mechanism. Based on individual security requirements a custom-designed RABC system can implement some or all of the RBAC components, which are basic RBAC, hierarchical RBAC, static separation of duty relations, and dynamic separation of duty relations. There is no doubt that RBAC will be a well-accepted access control mechanism in the near future.
CHAPTER 6
PROPOSED EXTENSIONS TO THE CURRENT NIST RBAC/WEB SYSTEM

In the year 2000, more than a decade after intensive research have been done in the development of RBAC mechanism as well as in design and implementation of different RBAC models, a standard for RBAC has been proposed [Fer00]. The proposed standard is an attempt to define fundamental components of RBAC and to serve as a foundation for product development, evaluation, and procurement specification [Fer00]. Until this date, no complete RBAC system has been developed yet. There is a RBAC package for Web servers developed by RBAC/Web team at NIST [Bark98], which includes most of the main components of RBAC concept such as roles and role hierarchies, role activation, constraints on user/role membership and role set activation inheritance, separation of duties (both static and dynamic separation of duties), using Perl, C, and CGI. The NIST RBAC/Web is a very advanced RABC system and we suggest an approach to extend it by using new techniques that would allow us to efficiently and effectively deploy the advantages of RBAC mechanism and incorporate them into the system. In particular, we introduce a maintenance tool for role-based access-control, which supports the administration and management of the system’s access control. So keeping all the components of the current NIST RBAC/Web system, we proposed to use a ready to use database server (Sybase) (instead of the RBAC/Web Database server) and SQL for manipulating RBAC data sets. The communication between the RBAC database and the system interface is provided by JDBC. In this thesis
our main focus is on the RBAC/Web API, where we implement basic administrative functions for a core RBAC model [Fer00] and add advanced review functions for administrative purposes, which are AssignedUsers, AssignedRoles, RolePermissions, UserPermissions, SessionRoles, and SessionPermissions. The rest of this chapter is divided as follows: in Section 6.1 we define the requirement specifications for a core RBAC model based on the proposed RBAC standard [Fer00]. Section 6.2 is dedicated to the BRAC/Web system’s design specifications. Section 6.3 specifies the system’s modules and Section 6.4 describes advanced review functions of RBAC/Web in our implementation. Section 6.4 and Section 6.5 list the assumptions that we use in our implementations and the rationale for our approach respectively.

6.1. Requirement Specifications for Core RBAC [FSGKC00]

The RBAC requirements specifications can be classified into three main categories:

- **Administrative functions** - creation and maintenance of elements sets and relations for building the various component RBAC models;

- **Supporting system functions** - functions that are required by the RBAC implementation to support the RBAC model constructs (e.g., RBAC session attributes and access decision logic) for user interaction with an IT system; and

- **Review functions** - review the results of the actions created by administrative functions.

6.1.1 Core RBAC System Administrative Functions

*Creation and maintenance of element sets:* The basic element sets in Core RBAC are USERS, ROLES, OPS (operations), and OBS (objects), where OPS and OBS are system-predefined. Administrators create and delete USERS and ROLES, and establish
relationships between roles and existing operations and objects. Required administrative functions for USERS are AddUser and DeleteUser, and for ROLES are AddRole and DeleteRole.

*Creation and maintenance of relations:* The two main relations of Core RBAC are user-to-role assignment relation (UA) and permission-to-role assignment relation (PA). Functions to create and delete instances of User-to-Role Assignment (UA) relations are AssignUser and DeleteUser. For Permission-to-Role Assignment (PA) the required functions are GrantPermission and RevokePermission.

### 6.1.2 Supporting System Functions

Supporting system functions are required for session management and in making access control decisions. When a user starts a session, a default set of active roles (a subset from the user’s assigned role set) is activated. During the session the user has an option to alter his session’s default set up by adding or deleting roles. Another option is a user can choose an active role set and then be authorized before activating the session.

The main system functions are as follows

- **CreateSession** - Creates a User Session and provides the user with a default set of active roles,
- **AddActiveRole** - Adds a role as an active role for the current session,
- **DropActiveRole** - Deletes a role from the active role set for the current session, and
- **CheckAccess** – Determines if the session subject has permission to perform the requested operation on an object.
6.1.3 Review Functions

The review functions for a core RBAC model return the contents of User-Role and Permission-to-Role Assignments. These review functions enable the system administrator to view all the users assigned to a given role as well as all the roles assigned to a given user. In addition, some session attributes – like the active roles in a given session, the total permission domain for a given session can also be determined by using review functions.

6.2 Design Specification

6.2.1 Authentication

RBAC/Web is used in conjunction with authentication and confidentiality services such as existing Web authentication and confidentiality mechanisms. These include username/password, Secure Socket Library (SSL), Secure HTTP (SHTTP), and Private Communication Technology Protocol (PCT). The Web server passes user identification information to RBAC/Web. It is the responsibility of the Web server to authenticate user and provide confidential data transmission as configured by the Web server administrator.

6.2.2 End User Scenario

Before access to a URL controlled by RBAC is permitted, end-users must establish an RBAC session. The end users can choose and/or are assigned a current active role set (ARS). The ARS determines the permitted operations that the end-user can perform on RBAC controlled URLs. The ARS remains in effect until the end-user establishes a new ARS. Generally an RBAC session requires an authenticated end-user. Figure 12 describes the use of RBAC/Web.
To establish a new session, the user chooses a set of active roles from the active role set choices presented to him by the system. After the session is established, the user’s access rights is the union of all the permissions assigned to the active role set of that particular session.

6.2.3 Authorization

Authorization is provided by the RBAC/Web CGI, which is derived from the W3 (i.e., CERN) Web Server. The RBAC/Web CGI uses its own set of configuration files. The RBAC/Web CGI uses the W3 style access control list files (RBAC_acl), which permit access control on a per file basis. The W3 ACL file is used to define which roles are permitted to perform which operation on which files. Each line of the W3 ACL file has the format:

```
<filename wildcard template> : <op list> : <role list>.
```

Users who have any of the roles in <role list> in their ARS may perform all methods in <op list> on each file which satisfies <filename wildcard template>.

The .RBAC_acl files reside on each server where RBAC/Web is installed.
6.3 System Modules

6.3.1 RBAC Database

The system’s RBAC database consists of data sets, which specify the relationship between users and roles, the role hierarchy, the constraints on user/role relationships, current active roles, and relationship between roles and operations. The data sets reside in Sybase database server and are created and manipulated by system manager. At the moment our database supports only a core RBAC system that meets the above mentioned requirements specifications.

- Associate with each role is a tuple $<\text{Role}_\text{name}, \text{Permissions}, \text{Cardinality}, \text{Assigned}\_\text{users}, \text{Static}\_\text{separation}>$. (In our implementation we do not consider the inheritance and dynamic_separation aspects of roles yet). Each role is assigned a unique name and an associated set of predefined access permissions. After a user is successfully authenticated to the system, he can start a session from a login program by choosing a set of roles from his assigned role set. The user’s access rights are the union of access rights of his current active role set (the roles activated in his current session). The cardinality attribute specifies the maximum number of users that can be assigned to that role. Assigned_users is a list of users, assigned to this role. And the static_separation indicates a set of roles, which share the static separation of duty relationship with this role.

- Associate with each user is a tuple $<\text{User}_\text{id}, \text{assigned}\_\text{roles}>$. Each user is assigned a unique user identification ($\text{User}_\text{id}$) and a set of roles assigned to the user by the system administrator.
• Each session is a tuple \(<\text{SessionId}, \text{Active role set}, \text{UserId}>\) associated with only one user. For each session an active role set, which is derived from the user’s assigned role set, is activated.

6.3.2 Administrative Module

The administrator performs a set of following operations:

• *Add a user to the users data set.* Error is returned if that user is already in the users data set.

• *Delete a user from the users data set.* Error is returned if that user has no entry in the data set or the user has roles assigned or has active role set.

• *Delete a user’s active role set.* Error is returned if that user has no active role set.

• *Assign roles to a user.* Add a new role to a user’s *Assigned_roles* data set. Error is returned if the user does not exist in the user’s data set, no role exists in the roles data set, the user has already assigned this role, or user has already been assigned a role, which has static separation of duties relationship with this role.

• *Remove a role from a user.* Error is returned if the user is not in users data set, no role exists in the roles data set, or this role is not in user’s *Assigned_roles* data set.

• *Add a role to the roles data set.* Error is returned if this role already has an entry in roles data set.

• *Delete a role from the roles data set.* Error is returned if this role is assigned to user(s) or no entry in the roles data set.

• *Set cardinality for a role to x.* Error is returned if no entry for role in roles data set, or number of users in role exceeds x.
- **Establish static separation of duty relationship** between role1 and role2 by adding role1 to role2 entry and role2 to role1 entry in the Static_separation data set. If either role1 or role2 does not have an entry in the Static_separation data set, create one. Error is returned if no entry for role1/role2 in the roles data set, or the static separation of duty relationship between role1 and role2 is already established.

- **Remove static separation of duty relationship** between role1 and role2 by removing role1 to role2 entry and role2 to role1 entry from the Static_separation data set. Error is returned if no entry for role1/role2 in the roles data set, or the static separation of duty relationship between role1 and role2 does not exist, i.e. role1 is not listed in role2’s entry and role2 is not listed in role1’s entry.

- **Assign a set of permitted operations to each role**. Error is returned if no entry for this role in the roles data set.

- **Remove a set of operations from a role**. Error is returned if no entry for the role in the roles data set.

### 6.3.3 System Administration API Module

The system administrator performs all of the above operations through system interface. The system interface interaction with the RBAC database is implemented in Java using JBDC (for database connectivity), Java servlets, HTML for creating the system’s interface.

### 6.4 Advanced Review for RBAC/Web System

Besides the two basic review functions (AssignedUsers(role_name), which returns the set of users assigned to a given role and AssignedRoles(UserId), which returns
the set of roles assigned to a given user), we implement the following advanced review functions:

- **RolePermissions (Role_name)** returns the set of permissions granted to a given role. Each permission is associated with an object and a list of operations that can be performed on this object. The function returns an error if the role is not a member of the roles data set.

- **UserPermissions (UserId)** returns the set of permissions a given user gets through his/her assigned roles. The function returns an error if the user is not a member of the users data set.

- **SessionRoles(SessionId)** returns the active roles associated with a session. The function returns an error if the session identifier is not a member of the session data set.

- **SessionPermissions(SessionId)** returns the set of permissions of the session, i.e., the permissions assigned to its active roles. The function returns error if the session identifier is not a member of the session data set.

These advanced review functions can help the system administrator in determining unauthorized accesses to system resources. For example being able to see the set of permissions assigned to each role and the set of permissions of each session together with user’s time stamp can help the system administrator to track down unauthorized users.

### 6.5 Assumptions

**Log in.** When a user logs in, the user is first authenticated (using UserId, password), then the user has an option of choosing an active role set for himself from his
Assigned _role set at login session.

_GrantPermission/RevokePermission_ functions grant/revoke one permission at a time.

6.6 Rationale

The proposed above improvement to the RBAC/Web model for WWW environment is justified by the following advantages:

- Since Java has been proved to be one of the most popular, flexible, and scalable object programming languages, and with all of the ready-to-use packages for network programming, database implementation, using Java for building system API is a very effective approach.

- The database server (Sybase) and a database query language (SQL) provide us all the necessary tools that can provide effective data query and solve complicated tasks to preserve database consistency that otherwise would require special tools.

- The communication between the system interface and the database can be nicely supported by ready-to-use tools (JDBC) for this purpose.

- We have provided a very friendly GUI that enables the system administrators to quickly and effectively perform their tasks.

- We have shown how some of the access control advanced functions can be implemented. These functions have never built before and they provide not only object-based but also user-based, role-based review functions that would be of great use for system security administrator. Being able to see which set of access permissions a particular user is granted through his assigned role set, the set of permissions granted to a particular role, the active roles/permissions associated
with a session can be of a great help in finding malicious activities and locating
the unauthorized user.
CHAPTER 7
CONCLUSION

From the level of interest in role-based access control shown by the research community and in the marketplace, it is clear that RBAC is becoming an integral part of the global information infrastructure [San00]. Today RBAC is becoming increasingly popular for commercial database and enterprise management, business-to-business and business-to-consumer e-commerce, and network operating systems. A number of important achievements have been done in the research arena; many aspects of RBAC models have been thoroughly studied and applied in various access control environments. The recently proposed uniform standard for RBAC by Ferraiolo, Sandhu, Gavrila, Kuhn, and Chandramouli serves as a common representation for access control models and policies, making it a suitable foundation for a policy coordination system as well as a foundation for product development, evaluation, and procurement specification. Despite a tremendous amount of work that has been done in this area, RBAC is still a rich and open-ended technology, which is evolving as users, researchers and vendors gain experience with its application [Fer00]. There are other aspects of RBAC that need to be developed to satisfy a wide range of access control requirements. Some of the open problems are:

- Developing a comprehensive RBAC administrative model for separating responsibilities in cross-organization systems, where user-role assignment can be handled by one organization while permission-role assignment is handled by
another [San01]. Furthermore new concepts in administration of RBAC have been introduced (delegation of access rights and roles, personalization, authorization of administration of RBAC, etc.) and need to be fully developed and added to future versions of RBAC standard.

- Due to the rapidly expanding nature of information technology, current concepts of RBAC must be developed further to match new applications requirements. For example assigning a role to a user will be considered as the job of not only some other user (or administrator) but it can also be any event-driven result such as a role is assigned to a user as a result of that user having won a contest in the Internet. At the present the delegation of access rights through role delegation has been considered in few studies and only at the developing theoretical role-based delegation models level [Bar00]. Implementing role delegation concepts is a challenging job and requires thorough system organization analyses.

- RBAC model can be extended to include new features such as introducing the notion of states [Ste01], sessions [Lee99], and information flow [Iza01] into RBAC models.

- One of the main components of RBAC, constraints, is also an interesting aspect for further research. Recently Gail and Ahn Ravi Sandhu [Ahn00c] have proposed the RCL2000 language for specifying RBAC constraints and have argued that prohibition and obligation constraints are both required with separation constraints being an example of prohibition. Since constraints play an important role in realization of any security policy, specification, and implementation of different kinds of generic and application specific constraints remains a
challenging task. At present only constraints on sets of roles and in particular on their ability to form user-role assignment relations have been implemented in RBAC products [Fer00].

- Separation of duties has been serving as an effective tool to enforce conflict of interest policies. But at the present the specification and implementation of static separation of duties (SSD) and dynamic separation of duties (DSD) remain an open field for deeper and broader investigation.

- The initial motivation for RBAC was on managing access rights in large-scale systems from a single organization’s point of view [San01], where the models rely on a single, per organization policy to define who undertakes each role, and what the relationships between roles are [Fer95]. At this stage of development RBAC should be considered for building security systems across multiple domains/ environments.

- The current RBAC standard, which focuses on defining fundamental components of RBAC [Fer00], is the first step in developing a complete and uniform standard RBAC API that would in turn promote the development of innovative authorization management tools by guaranteeing interoperability and portability.


BIOGRAPHICAL SKETCH

Diep Thi Ngoc Nguyen was born in Vietnam and she is a permanent resident of the United States of America. She majored in linguistics and foreign languages and got a master’s degree in interpretation; she speaks Vietnamese, English, Russian and German. Her current research interests include computer network and security. She is also interested in distributed database management. She is expecting to receive her Master of Science degree in computer engineering from the University of Florida, Gainesville, Florida, in December 2001.